AR-009-950

DSTO-TR-0460

The SPRing Approach to Software Costing

Gina Kingston and Martin Burke

19970624 096

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

The SPRing Approach to Software Costing

Gina Kingston and Martin Burke

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0460

ABSTRACT

This paper describes a two-phase approach to Software Costing which has been proposed by the iMAPS Software Costing Research team. The proposed approach consists of obtaining an initial, rough, estimate of the Cost of the system during the first phase. During the second phase the estimate is refined throughout the development of the system as more information becomes available.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108

Telephone: (08) 259 5555
Fax: (08) 259 6567

AR No. AR-009-950
December 1996

APPROVED FOR PUBLIC RELEASE

The SPRing approach to Software Costing

Executive Summary

Software is an increasingly important and costly component of many Defence systems. Methods for determining the Development Costs of such systems tend to be inaccurate and imprecise. This document outlines a two-phase approach to Software Costing which is being developed by the iMAPS Software Costing Research team and explains how the approach was designed to reduce the risks to all parties involved in Software Acquisition.

The method consists of producing a coarse estimate of the Software Cost and then continually refining it. The first, or Slicing phase produces a coarse estimate because little information is available early in a Software Development. It is obtained from the Size of the software (its Capacity) and the effectiveness of the development environment (its Difficulty). Other Cost Estimation techniques produce a single value which is often used as an (unrealistic) target for the Software Cost. In the SPRing approach, the Slicing phase produces a range estimate which is refined during the Progressive Refinement phase as more information becomes available.

Future work under the SCARAB task will investigate these ideas further and determine their effectiveness.

Authors

Gina Kingston
Information Technology Division

Gina has been employed in the Software Engineering Group of the Information Technology Division of the Defence Science Technology Organisation (DSTO) since graduating from the University of Tasmania with a BSc with First Class Honours in Mathematics in 1990. She is currently undertaking a PhD in Software Reviews through the University of New South Wales' School of Information Systems and working on the Software Cost and Risk Benefit Analysis (SCARAB) task at DSTO

Martin Burke
Information Technology Division

Martin has a BSc(Hons) in Physics, a MSc in Mathematical Statistics, and a PhD in Engineering Mathematics. He has held scientific and management positions at Rolls Royce (Aero), the SEMA Group Research Centre and the UK Atomic Energy Authority. Martin joined DSTO in 1991 as a section leader in Software Engineering Group and is the Task Manager of the integrated Measurement, Assessment and Prediction of Software (iMAPS) task. His current fields of specialisation include: Software Cost Prediction, Software Risk Assessment, and Safety Critical Systems and Software.

Contents

1. INTRODUCTION

Objectives

This document outlines a new approach to Software Costing being developed by the iMAPS Software Costing Research team. It provides reference material which will be used in other publications by the iMAPS Software Costing Research team.

Context

This work forms part of the DSTO iMAPS Task DST 93/349 [Burke, 1995], a 3 year DSTO task which aims to provide an integrated approach to the description, measurement, assessment and prediction of software attributes.

Research on the Software Cost Prediction foci of the iMAPS task started in mid-1994. An initial statistical investigation using public domain data has already been undertaken [Kingston et al., 1995] through a Co-operative Education Enterprise Development (CEED) agreement [Kiermeier, 1994] with the University of Adelaide. Current work is focusing on the development, evaluation and refinement of a new model for Software Development Cost Prediction and will contribute towards Gina Kingston's PhD studies at the School of Information Systems, University of New South Wales.

Motivation

Software is an increasingly important and costly component of many Defence systems. Methods for determining the Development Costs of such systems tend to be inaccurate and imprecise. As a consequence, the software acquisition process tends to be high risk for both the Australian Defence Organisation (ADO) and the industries involved. The iMAPS Software Costing research aims to reduce the risks for all parties by providing a suitable approach to Cost prediction. The approach will reduce risks by being developed on data relevant to Defence. It will further help in the risk management of projects by making the risks more amenable to quantitative, repeatable and auditable analysis through the use of cost-based risk intervals.

Assumptions

The Cost associated with a software product has many elements including: the cost of the hardware it will be developed on and, Software Development and Maintenance costs. The iMAPS Costing Research will focus on just the Software Development Cost. Development Cost is assumed to be primarily due to the cost of labour and may be thought of as Development Effort multiplied by the average cost per unit Effort.

Related Documents

A planned series of documents will focus on the concepts introduced in this paper. Particular attention will be paid to measurement theoretic considerations. These documents include:

"iMAPS: Capacity - A New Measure of Software Scope for Effort Prediction "

"iMAPS: Capturing the Effect of the Work Environment on Software Development Effort through Difficulty" and

"iMAPS: Methods for Progressively Refining Development Effort Predictions".

These concepts will be explored through the statistical analysis of Defence data, which is currently being collected. A large data set is required to ensure the validity of results. Parties which may be interested in supplying data can obtain a copy of "iMAPS: Collecting Data for Software Costing" [Kingston, 1996], a document describing and motivating the collection procedure, from the authors or from Peter Fisher of DSTO's Software Engineering Group. He can be contacted by email at: "Peter.Fisher@dsto.defence.gov.au". Details of the analysis process will also be published.

The iMAPS Software Costing Research team are also investigating existing approaches to Software Costing and will publish reviews of their findings. The first such document is "iMAPS: A Review of Software Sizing for Effort Estimation" [Kingston et al., 1996b].

Intended Readership

This document presents a high-level, introductory view to the iMAPS team's approach to Software Costing. As such, it may be of interest to ITD management, other members of DSTO's Software Engineering Group and ITD's C3I Systems Engineering Group. It may also be of interest to other ADO personnel involved in the acquisition of software-intensive systems. In addition it may be of interest to researchers in Software Cost Estimation and practicing software engineers, particularly those in the Defence community.

Layout

Section 2 provides background to Software Development which is intended for those who are unfamiliar with this topic.

Section 2 also introduces some Costing terminology which is used throughout this, and other papers on Software Costing by the authors.

Those readers interested in an overview of the proposed approach to Software Costing should read Section 3. Readers interested in future plans and a summary of the potential benefits of the approach should read the Discussion in Section 4.

2. BACKGROUND

A substantial amount of literature is available on Software Cost Explanation techniques dating from the 1960's. (See [Kemerer, 1991] for a summary of approaches commonly used.) Most of this work relates some measure of Software Size with Effort. While the correlation between Cost and Size is statistically significant [Kitchenham, 1992], it is not strong enough to provide useful Cost Estimates. Therefore, adjustment factors are usually added and linear regression is reapplied to try to improve the model. In addition, most of the work focuses on Effort Explanation, rather than Effort Estimation or Prediction (See Section 2.2 for definitions of these terms).

Literature surveys on Software Costing and Software Sizing Techniques are currently being conducted by the iMAPS team, and papers presenting the results are being prepared [Kingston et al., 1996b;Kingston et al., 1996a]. The conclusions so far are that the results given by the current state-of-the-art techniques are still of limited accuracy and rarely attempt to quantify precision in predictions. Correlation efforts [Kitchenham, 1992;Kingston et al., 1995] have shown that measures of Software Size have the greatest correlation with Software Cost. They also show that most of the adjustment factors used in current models have little or no correlation with Development Effort. The only adjustment factors which appear to have statistical significance are the Development Environment and language.

Section 2.1 provides background information for readers who are not familiar with why it is difficult to develop Software. This should help these readers understand why Software Developments need to be risk managed: it should be read prior to Section 3.2 which describes the limitations of current approaches to Software Costing.

Section 2.2 explains some Software Costing terminology used in this, and related, papers. It includes explanations of the differences between the terms Estimation, Explanation and Prediction.

Section 2.3 discusses some preliminary investigations which offer support for, and insight into the methods of Software Costing proposed in this paper.

2.1 "Nature of" Software Development

A description of a simple software development process can be found in [Conte et al., 1986]. However, perhaps the best, and most often cited, reference on the nature of software development is "No Silver Bullet: Essence and Accidents of Software Engineering", [Brooks, 1987]. In it, Brooks lists four properties of software

development. These are: Complexity, Conformity, Changeability and Invisibility. One more property has been identified by the authors: Novelty.

Complexity. Complexity in software arises from the interaction between the components of a software system, and the interaction of the development team. Software systems are becoming increasingly large and sophisticated. As the size of the system increases, the potential for intended and unintended interactions between its components increases exponentially. Also, as the sophistication of the system increases, the specialisation of the developers increases. It is not uncommon for User Interface Experts, Business Domain Experts, and Software Engineers to work side by side in the development of modern systems. The complexity of modern systems is particularly noticeable when the project staff change, either due to staff turnover, or when the product moves from a development to a maintenance phase.

Conformity. Software is considered to be a very flexible medium. As such, it is expected to conform to its environment. This includes conforming to the constraints of the hardware on which it runs, conforming to the manual procedures in place prior to automation and conforming to the desires of its operators.

Changeability. Unlike most other manufactured goods, the most successful software applications are those which are subject to change. A successful application outlives the hardware on which it was developed to run, and the users of a successful application make it an integral part of their work style, placing demands on the software which it was never intended to meet. Other applications may also be subjected to pressures for change. For example, when the requirements of the user change during the development of a system, there is often pressure to change the software because it is seen as the easiest component of the system to change.

Invisibility. Software is difficult to visualise, even for experts in the field. (Brooks states that it is inherently invisible. However, the authors believe that work such as [Baker and Eick, 1993; Eick et al, 1992; Vernik et al., 1991; Vernik et al., 1993; Vernik, 1996] is making progress in this area.) This lack of visibility makes it difficult to check the correctness of programs and their specifications, and to maintain and change programs. The lack of visibility to application users can also result in an incomplete understanding of the capabilities and limitations of computers and subsequent placement of unreasonable demands on the software.

Novelty. One feature which distinguishes the production of software from many other industrial engineering activities is that the process of obtaining many identical copies of a system is extremely easy. The effort which goes into developing software is all associated with the original item. Consequently each software development activity is involved in the production of something new. While often the product may be similar to those previously developed, it is never identical.

Programmable computers are a relatively recent invention which have emerged into prominence in the last 50 years. Computing technology (including software and the

techniques used to develop it) has changed, and continues to change at an increasingly rapid rate. Software is not only expected to perform an increasing variety of functions, but it is expected to make use of the latest technology. Thus, developing a product similar to an existing one can be a novel experience, if the latest tools and techniques are used. Rapid changes to technology affect the feasability of projects and the economic advantages offered by new technologies mean that whole systems are often updated. In this environment new products similar to those produced in the past, are common.

2.2 Estimation, Explanation and Prediction

The terms Estimation, Explanation and Prediction are often used inconsistently and incorrectly in the Software Costing field. This section provides definitions of these and related terms as they are used throughout this and related iMAPS documents. Figure 1 depicts the relationships between the terms.

Definition: Costing

Costing is the generic term used to describe all work on developing models for Software Development Costs, regardless of their use, testing, or how they were developed. In other contexts it may also be used to refer to models of Software Maintenance Costs.

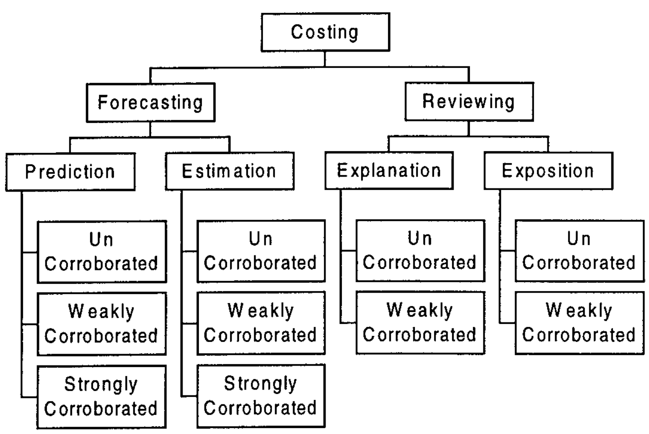

Figure 1: Costing Terminology

Definition: Reviewing

The first stage in developing a Software Development Costing model is normally to explore potential models using historical data (on development Effort, Size and Productivity Factors). Models which use data obtained after project completion will be termed Review models.

There are two types of review models: Explanation and Exposition. Explanation models are the simpler.

Definition: Explanation

Explanation models are the first (stage in) models developed to explain the behaviour of past projects. They contain estimates of their accuracy; that is, they have been checked for biases. For this reason, most models which claim to be Estimation models, would be more correctly termed Explanation models.

Definition: Exposition

An Exposition model is an Explanation model that has been enhanced by a mechanism for determining the precision of the model, such as Prediction Intervals [Matson et al., 1994]. (Prediction Intervals provide upper and lower bounds to the development Effort. Prediction Intervals are determined by fixing the probability of the actual Effort lies outside the Interval and then determining the location of the bounds. A prediction model, with Prediction Intervals shown, is given in Figure 3.)

Definition: Forecasting

Unlike Review models, which are retrospective, those which are to be used for Costing a project during its Development should be developed from Costing data obtained before and/or during the Development.

There are two types of such Forecast models: Estimation and Prediction models. Estimation models are the simpler.

Definition: Estimation

A model is termed an Estimation model if it is a Forecast model whose accuracy has been determined.

Definition: Prediction

A model is termed a Prediction model if it is an Estimation model for which Prediction Intervals [Matson et al., 1994] have been determined.

The validity of Software Costing models may be checked (corroborated) using a variety of techniques and this method can be used to further classify the models. Models are only considered to be corroborated if the data which was not used in the development of the model is used to check the model.

Definition: Un-Corroborated

Un-corroborated models have their accuracy (and precision) checked for biases using at most the information used to develop the model. Checking techniques may

include statistical checking such as statistical checks for differences between models, Mean Residual Error checks and heteroscedastity checks (tests for changes in error with Size). All types of Costing models may be Un-Corroborated.

Definition: Weakly Corroborated

When Weakly Corroborated models are developed, a randomly chosen subset of the available data is set aside for testing. After the model has been developed from the remaining data, from one or more projects, Effort estimates are obtained from the test data and compared to the Actual Effort. Accuracy is measured by checking for biases in these estimates. Precision can be determined from the absolute errors in the estimates. All types of Costing models may be Weakly-Corroborated.

Definition: Strongly Corroborated

Strongly Corroborated models are similar to Weakly Corroborated models, in that the model is checked using data that was not used for the development of the model. The difference is that Strongly Corroborated models are tested on new projects where the estimates are used during the development process, not just on projects for which data was initially collected. Therefore, only Forecast models can be Strongly Corroborated. Models may be Strongly Corroborated to check against biases in the initial data collection, and biases introduced by the estimation process. For example, the development of many of the models in the literature used only data from successful (or at least complete) projects.

2.3 Preliminary Investigations

Early studies into Software Costing were conducted under a CEED (Co-operative Education for Enterprise Development) agreement [Kiermeier, 1994] between the University of Adelaide's Statistics Department (UASD) and the Defence Science and Technology Organisation's Information Technology Division (ITD).

This work was based around a statistical analysis performed on a collection of public domain cost estimation data containing 371 data points. The data contained information on Effort, Size - in Function Points (FP) or in Lines of Code (LOC), Duration and a variety of Productivity Factors. The analysis focused on the use of prediction intervals and local data as well as investigating the significance of individual productivity factors. Some of the results of this work are documented in [Kingston et al., 1995] while the complete results are only published in a Commercial-In-Confidence document.

This work, together with previous experiences of the authors, offers some insight and support for the iMAPS approach proposed in this paper.

2.3.1 Local Data and Difficulty

One of the main areas on which the CEED analyses focused was the impact of Local Data. These analyses (see Figure 2) showed that there were differences between the different data sources and showed that incorporating the data source offered statistically significant improvements.

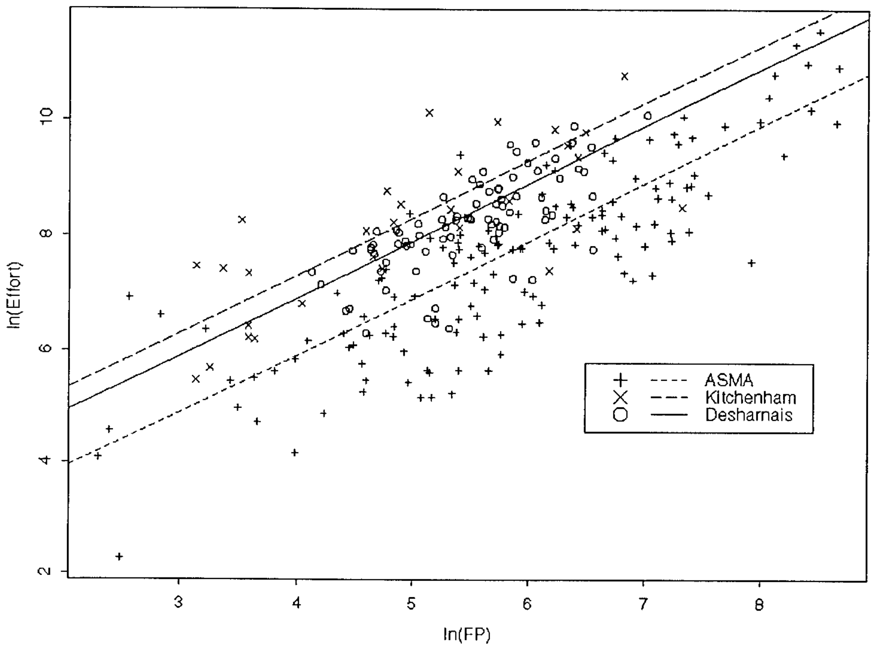

Figure 2: Ln(Effort) versus Ln(Function Points) according to data source. The data sources are ASMA, Kitchenham and Desharnais

However, only the coefficient of the equation (of the form given in Equation 1), and not the exponent, varied with the data source. Several researchers in Software Costing, have previously suggested that the exponent would also vary with the data source. This is the primary motivation for the proposal of the Difficulty concept (see Section 4.4). Difficulty is a factor, similar to c of Equation 1, which depends on the development environment. Calibrating a Software Costing model may be one way of determining the Difficulty for an organisation which regularly develops the same type of products.

$$Effort = c\ Size^{k}$$

Equation 1: Effort Relationship

This phenomenon also means that only a relatively small amount of data is required to calibrate models with local data, since only the value of c, and not that of k, needs to be determined from the data.

2.3.2 Slicing

When plotted on a ln-ln graph, (where ln is the natural logarithm) the differences in these models due to the impact of the local data show themselves as parallel lines which appear to divide the data into bands or 'slices' (See Figure 2 which shows Effort versus Function Points (FP) for three different data sources). This provides some support for the Slicing approach described in Section 4.2.

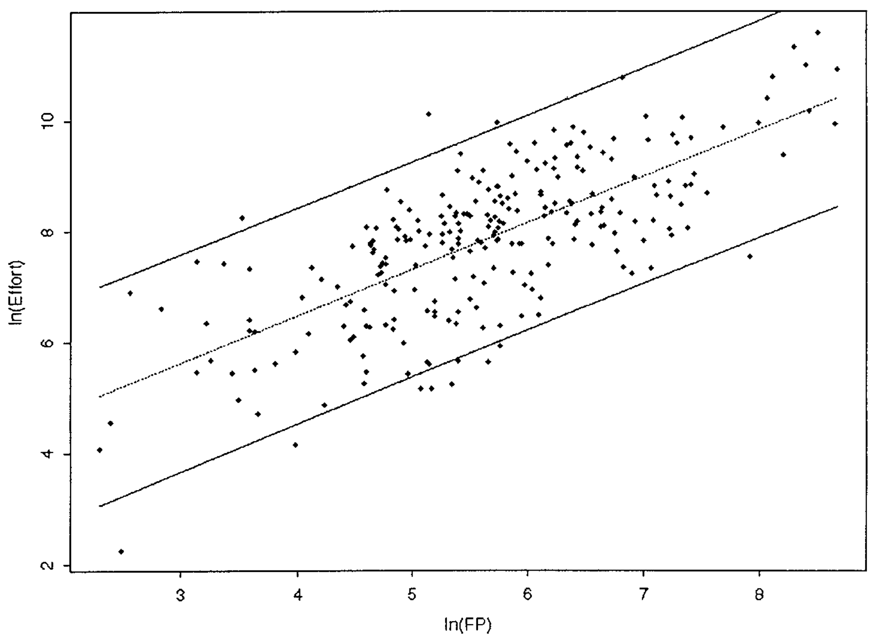

Figure 3: Prediction Intervals on the ln-ln scale.

Intuitively, we suspect that a variety of different breadths of slice will exist: with factors such as Organisations being associated with broad slices, factors such as Development Environments (or Difficulty, see Section 4.4) being associated with narrower slices.

2.3.3 Prediction Intervals and Progressive Refinement

The CEED analyses also produced Prediction Intervals for Effort. One of these is shown in Figure 3. The middle line in this figure is the predicted ln(Effort) and the prediction interval is given by the two outer lines. This interval is obviously very wide,

meaning there is a high risk that Cost overruns or underruns are likely if a single Cost prediction is made.

Observation of this phenomenon motivated the proposal of the Progressive Refinement (see Section 4.5) approach where Costs are tracked and re-estimated throughout the project.

2.3.4 Quantisation

The authors' observations of Software Engineers, and people in general, have cast some insights into how people naturally proceed in estimation tasks. Initial estimates tend to be made based on a coarse analysis of the problem, often performed by partitioning known information into large groups of "similar" information. Often this similar information is that the values of particular pieces of information, or measures, lie within broad ranges. This motivated the consideration of quantisation in the SPRing approach presented in this paper.

2.3.5 Early Size Measures and Risk Management

One surprising result from the CEED studies was that, while the Prediction bands were wider for models based on Function Points than for models based on Lines of Code, the upper bounds were similar percentages of the average value (once the model has been adjusted for biases). This means that the risks associated with Cost *overruns* were similar for the two types of models. From this result, we can argue for the use of Size Measures which can be determined early in a software development. Such measures can not only be determined early, but Effort models (based on at least some of these measures) have the same associated risk (upper bounds vs predicted values) as models based on other measures, particularly Lines of Code, which cannot be determined until late in the development. This supports the use of an early Size measure, Capacity - see Section 4.3, in the iMAPS team's SPRing approach to Software Costing.

2.3.6 Productivity Factors and Difficulty

The CEED studies investigated the impact of 29 Productivity Factors. In isolation, few of these had significant impact. Those which did, particularly Programming Language Level, will be considered in the refinement of the Difficulty concept.

3. AIMS

3.1 Practitioners Requirements

The Software Engineering Institute (SEI) conducted a survey to determine the current state of Software Costing in practice, and the improvements in Software Costing desired by software practitioners [Park et al., 1994].

The SEI is operated by Carnegie Mellon University and sponsored by the US Department of Defense. Thus, the survey was widely distributed within government, industrial, and academic circles. Responses were received from 81 Government/Military, 159 Industry, 4 Academia, and 5 Unknown sources.

The SEI concluded that software practitioners had a strong desire for improved software (Cost, Schedule and Size) estimation. Comments on the improvements which the practitioners thought would help included:

- Processes rather than tools should be developed. The processes should enable feedback at milestones during the development. Statistics should be used for making future estimates.

- Better mechanisms for capturing historical data for comparative purposes and the re-calibration of models for organisational use should to be developed.

- Better models for Software Sizing than Lines Of Code (LOC) and Function Points (FP), which enable Size to be measured from the requirements, need to be developed. The metrics need to be suitable for new types of development environments. For example, object oriented development environments.

While this survey was conducted in the USA, the authors believe that these concerns are also held by the Australian Defence Organisation and will check their beliefs by conducting a relevant, local survey.

3.2 Limitations of Current Approaches

Current approaches to Software Costing provide a poor basis for risk managing a project.

Most of the current approaches provide a single point estimate of Cost, which gives no indication of the probability of other values occurring. The ranges of estimates, for the approaches that provide ranges for a given probability of occurrence, are very large [Kingston et al., 1995].

Most of the approaches fall into one of two categories. The approaches in the first category provide a rough estimate of Effort based on information that is available early in Software Development. Those in the second category are based on information that is only available late in the development. Effort estimates made using these (second category) approaches are thus not available until well after project planning and budgeting. Few approaches use (combinations of) information that can be used to track progress throughout the project. Furthermore, the authors know of no approach which can relate later estimates to those made earlier in the project.

Finally, many of the current approaches to Software Costing are not statistically sound in a rigorous mathematical sense. This means that there is little reason to believe that they will apply to projects other than those on which they were originally developed.

3.3 Objectives

One of the long term goals of the iMAPS Task is to determine a robust approach to software cost estimation which can be exploited throughout the Australian Defence Organisation (ADO). The objectives of this approach are:

(a) to enable software cost estimates to be made, and subsequently refined, at all stages of software development from feasibility studies to deployment;

(b) to quantify the uncertainty involved in software cost estimates in a way which enables rational, risk-management of software development;

(c) to be easily understood and used by Defence project staff who are not necessarily software engineering specialists;

(d) to be neither technology nor application specific.

4. THE SPRing APPROACH

The new approach to Software Costing being developed by the iMAPS team is called the SPRing approach. The name SPRing was chosen because the approach combines their Slicing (S) and Progressive Refinement (PR) approaches and thus applying them may be called SPR-ing, or SPRing.

4.1 SPRing: A Two Phase Approach

The cost of developing a new software system cannot be determined precisely early in its development. However, as more information becomes available during the development, the costs can be increasingly precisely and accurately determined. Fortunately, in most circumstances cost estimates required early during the software development are not required to be precise. Increasingly precise and accurate estimates are required as development progresses.

A two phase approach to Software Cost Prediction is conjectured. The first phase is intended to provide a coarse-grained prediction in the Software Development, such as at the feasibility analysis stage. The second phase is intended to be an iterative process where this estimate could be refined as the Development proceeds and additional information becomes available.

The approach should lend itself to the support of a rational approach to the risk management of software development projects. It should enable an inexpensive

estimate of the Software Cost to be made quickly with minimal information. This estimate could be used to determine if the project is financially feasible, and could allow alternative development strategies to be considered, before a significant commitment is made to the project. If the project appears viable, additional time and money could be invested to obtain increasingly accurate and precise estimates.

Furthermore, it could enable the progress of the development to be tracked, and changes in the development to be captured by the model. This would enable increasingly tight Cost control over projects as their development progress and budget and schedule constraints become more severe.

This document outlines each of the concepts of the approach and how they relate to each other. Each of the concepts will be documented separately in an ESRL Technical Note or Technical Report. These documents will discuss how the concepts relate to similar topics in the literature, details of their foundations, how they are used, how they will be analysed and will identify their strengths and limitations.

4.2 Slicing

Slicing is conjectured as the method of determining a coarse-grained prediction of Cost as required for the first phase of the SPRing approach (See Section 4.1). The proposed Slicing method could be used early in the Software Development to provide a rough estimate of the cost during the feasibility analysis of the project.

The Slicing method is based on the main conjecture of the iMAPS research team: that Effort is a function of Capacity (See Section 4.3) and Difficulty (See Section 4.4). Capacity is a measure of Software Size and Difficulty is a measure of the suitability of the Development Environment. Slicing is intended to provide Effort ranges for broad bands of Capacity and Difficulty. While the approach could be accessed through tables and equations, it is likely that its use would be facilitated by the graphs of the form shown in Figure 4 and Figure 5. (See below for an explanation.)

The Slicing approach is intended to have the following features:

- Quick and easy to use.

- Intuitively appealing to cost estimators.

- Could determine Cost estimates as intervals or quanta. (The motivation for quanta is discussed in Section 2.3.4).

- Could be used early during Software Development and therefore can be used when determining the financial viability of a project.

- Could be used in several ways, including investigation of Cost, Size and Environmental trade-offs (see Figure 4 and Figure 5 and the following explanation).

- The Cost component of interest is assumed to be correlated with the development Effort.

- Assumes that Capacity and Difficulty are the two main factors which determine Software Development Costs.

- Could be used as an input to the Progressive Refinement approach which enables more precise estimates to be made as the Software Development progresses.

Graphs of the form shown in Figure 4 and Figure 5 could be developed to relate Capacity and Difficulty to Cost (Effort). Such graphs would be based on the same formula, which would be derived using the application of statistics to a large Defence data set. This formula would also be available to allow more accurate costs and prediction intervals to be determined. The formula is expected to be of the form given in Equation 2. This form is based on knowledge of other cost models [Kingston et al., 1995] which show Effort to be proportional to Size, and the expected impact of Difficulty (see Section 4.4). (Note that if this relationship is found to be non-linear then transformations can be applied to Capacity and Difficulty to obtain a linear relationship.)

$$E = C \times D$$

Equation 2: Equation for Estimating Effort (E) from Capacity (C) and Difficulty (D).

The graphs shown in Figure 4 and Figure 5 are provided to show different perspectives of the formula given in Equation 2.

The graph in Figure 4 clearly shows the quantisation of Difficulty, but it is not clear that Capacity is also quantised (as lines connect the distinct Capacity values). A graph of this form could be used to investigate the effect which changes in the Development Environment (and Difficulty) would have on Cost. A single (Difficulty) line from such a graph could also be used when the Difficulty is known, but there is some flexibility in the Capacity to be delivered. It is likely that organisations customising Slicing for their own Development Environments could use a graph of the form shown in Figure 4.

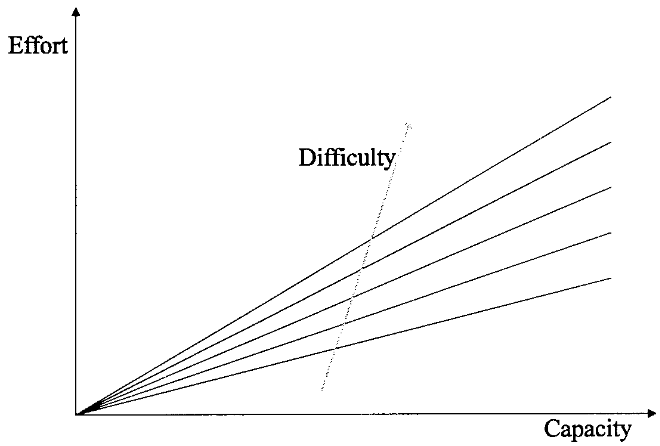

Figure 4: Effort Estimates Changing with Difficulty

In Figure 5 the quantisation of both Capacity and Difficulty is clear and Cost also appears to be quantised. However, while quanta might be determined from the iMAPS investigations, it is likely that the Effort (or Cost) will not be quantised prior to the initial investigations. A graph of the form shown in this figure could be used in the same manner as those of the form shown in Figure 4. However, it could be used more readily than the graphs of the form shown in Figure 4 to investigate Capacity versus Difficulty trade-offs in Fixed-Price contracts.

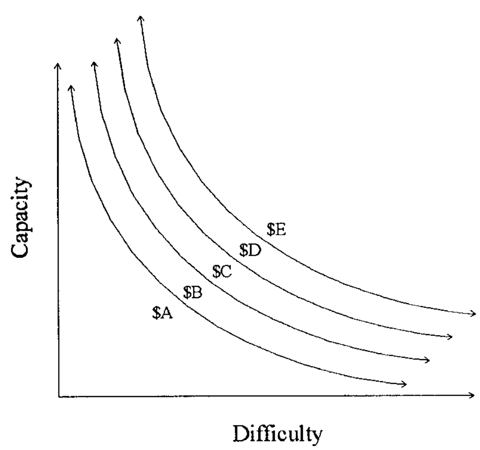

Figure 5: Slicing Model

The Slices, as shown in Figure 4 and Figure 5, or as obtained using a formula of the form given in Equation 2, could be used in a number of ways to investigate Cost, Size (Capacity), and Development Environment (Difficulty) trade-offs:

- To determine the local Difficulty given project(s) of known Cost and Capacity.
- To determine the Capacity that can be delivered for a fixed Cost given a particular Difficulty.

- To determine how much reduction in the Difficulty is required to achieve a given Capacity for a given Cost and;

- To determine the Cost of Software Development with a given Capacity and a given Difficulty.

4.3 Capacity

Capacity is proposed as a new candidate Software Size measure. It was conjectured after considering the benefits and limitations of existing measures of Software Size. Capacity is intended as a measure of Size that could be used, and be relevant, both before and after a Software System is developed. Capacity is only intended to capture what a software system does (or will do) and does not consider how it does (or will do) it. Capacity is intended to look at what the software directs the computer to do. It is asserted that computers can perform two main activities - communicating with external devices and processing information (both symbolic and numeric). Software can make computers appear "clever" by making them do many of these types of activities quickly. Capacity is intended as a measure of how much the software makes the computer do - that is, how much of communicating and processing the software controls.

Definition (Preliminary): Capacity

The Capacity, C, of a Software System is a measure of its Size or Functionality. It is defined as an increasing function of the number of Basic Manipulations (BM) which must be performed by the Software System to deliver its functionality.

A Basic Manipulation is either a Basic Data Transfer (communication) or a Basic Data Transformation (processing). It is not expected that these Basic Manipulations would be measured directly. Instead, indirect measures would be developed which identify standard combinations of Basic Manipulations that could be counted more easily than individual Basic Manipulations.

It might be possible to measure the number of Basic Manipulations by counting collections of them. However, in practice, Capacity might be evaluated by comparison to existing systems, or standards, of known Capacity.

Capacity is intended to have very different properties to the current measures of Size. These intended properties include that:

- Capacity could be naturally quantised in broad categories or be used as a fine-grained measure suitable for more detailed Software Sizing and Costing. (The motivation for quantisation is discussed in Section 2.3.4.)

- Capacity could be used without fine-grained knowledge of the project.

- Capacity could be measured very early in the Software Development - after the initial requirements have been captured, and could be refined later in the development.
- Capacity would be applicable for all Application Domains.
- Capacity would be independent of the environment in which the system will be developed.
- Capacity could be determined quickly and simply.

The main hypothesis of the iMAPS Costing Team is that Software Capacity and Difficulty (see Section 4.4) would correlate well with Cost quanta.

4.4 Difficulty

A large number of fine grained adjustment factors have been proposed as modifiers for Software Cost estimates. (For example, 29 productivity factors are analysed in [Kingston et al., 1995]). They are typically applied by giving a ranking to the influence of the factor (eg on a scale of 1 to 5), combining the factors using a weighted arithmetic sum, and multiplying the Size by the resultant number (the c of Equation 1). This approach has several problems including:

- Most of the factors are not statistically significant. (According to [Kitchenham, 1992] and [Kingston et al., 1995] the only currently considered adjustment factors which are significant are: Development Environment and programming language.)
- The determination of the ranking is subjective.
- The factors are not independent but, by using a weighted arithmetic sum, they are treated as if they are.

However, Software Size alone does not correlate well with Software Cost [Jeffery and Low, 1990]. Boehm's Basic COCOMO relates Cost to Size using an exponential relationship as shown in Equation 1 [Boehm, 1984]. However, when used on the 63 project data set on which it was developed, it is said to be accurate to within a factor of 2 only 60% of the time [Heemstra, 1992]. It is suspected that the results could be significantly worse on other data sets.

Difficulty

We have conjectured that there are two factors that could be determined early in the Software Development, which could be used to predict the Development Cost. The first is the proposed measure of Software Size that we call Capacity. The other factor, called Difficulty, is proposed as a measure of the effect the Development Environment and product constraints (see *Product* in this section) have on the ease of developing the software system. It is hypothesised that these factors could be used in the Slicing

Method (see Section 4.2) to obtain a coarse Prediction of the Software Development Effort. The Prediction could then be refined (see Section 4.5) to obtain more precise estimates later in the Software Development as project specific information becomes available.

Difficulty was proposed to overcome some of the problems with fine grained measures [Bailey and Basili, 1981] which appear to have little influence on software cost [Kitchenham, 1992]. The authors believe that one reason for this problem is that the fine-grained measures are highly inter-dependent.

Definition (Preliminary): Difficulty

The Difficulty, D, of a Software Development is defined as a measure of the effect the Development Environment and product constraints (see *Product* in this section) have on the ease of developing the software system.

It is suspected that there are a number of factors which determine Difficulty. We may never know what all of these factors are, but it may be possible to use a subset of these to determine Difficulty to the accuracy and precision required for a particular purpose. It is preliminarily assumed that the three most important factors for determining Difficulty are: Process, Product and Resource. (These are described below.) The following example (which uses the terminology introduced below) shows that we cannot assume that the three factors are independent.

Example:

1. The development of a simple product (where errors are not very likely) tends to cost *less* when there are less "checks" in the process.

2. The development of a complex product tends to cost *more* when there are less "checks" in the process (because it tends to cost more to correct errors found late in the development process).

Thus, an increase in the number of "checks" in the process may either increase or decrease the Cost of the project depending on the complexity of the product.

The authors suspect that, if one organisation does similar sorts of projects then the Difficulty for their development environments, and hence their projects, should remain relatively constant.

It is suspected that Difficulty is a complex function which, for the purpose of obtaining coarse Software Development Cost Predictions, may never require detailed investigation. Therefore, an approximation will always be required. One commonly used approximation technique is to consider functions as a weighted arithmetic sum of their inputs. From the example above, it can be seen that this approach is not sufficient for Difficulty. One approach which could be taken involves two steps:

- Quantising Difficulty, so that it could only take a discrete range of values.
- Using a "look-up" table to define the mapping from Process, Product and Resource to Difficulty.

This would also allow Process, Product and Resource to be quantised and could allow Difficulty to be easily assessed with a minimum of information.

Process

The Process attribute is intended as a coarse measure which captures the degree of control and rigour of the method used to develop the process. It is provisionally assumed that Process can take three values:

- Ad hoc: Poorly defined and controlled process
- Controlled: Well defined process, with some control
- Intensive: Well defined process, with strict controls

Things that should be taken into account when considering the rigour of a process include: the method used to handle changes to the requirements, what information is measured and how it is recorded and used, and the independence and nature of software evaluations.

Product

The Product attribute is intended as a measure of the demands and constraints placed on the product due to the environment in which it is to be developed, maintained and operated. Constraints include things such as: Availability, Reliability, Maintainability, Safety, Security, Real-Time, Storage and Timing requirements. It is provisionally assumed that there are three grades of Product:

- Easy: No constraints or demands
- Intermediate: A few compatible constraints or demands
- Difficult: Many or conflicting constraints and demands

Resource

The Resource attribute is intended as a measure of the extent and availability of appropriate human, financial, temporal, and computing assets during the project Development. Resource may also be considered a quantised attribute. The description given below gives the coarsest possible quantisation.

- Ample: The majority of resources are readily available at a suitable or better level. One possible set of criteria for Ample Resources would be that there is sufficient Time and Money, and suitable People to complete the project. Additional resources might also be required such as two or more of: language support, tool support, suitable platform and suitable reuse library.

- Constrained: The majority of resources are at a low level of availability or suitability.

4.5 Progressive Refinement

Progressive Refinement is the conjectured method for refining Cost Predictions. It is the second phase of a two phase approach to Cost Prediction that commences with Slicing (see Section 4.2). It would allow Cost Predictions to be refined by incorporating:

- updated information (eg Effort estimates from the Slicing approach which were modified due to changes in the scope (Capacity or Difficulty) of the project),

- more detail on existing types of Cost information (eg Effort estimates based on actual rather than estimated Lines of Code),

- new types of Cost information (eg Effort estimates based on Function Points and Lines Of Code) and actual Costs for the Development stages completed.

It would use methods for:

- refining estimates

- identifying patterns in changing estimates

- identifying high-risk situations where either the refinements don't conform to an existing pattern or the requirements conform to a pattern of escalating Costs etc, and

- determining the precision of the estimates in all these circumstances.

Progressive Refinement would start with an initial imprecise estimate and then, during the Development of the software, this estimate would be refined. While the details of the Progressive Refinement approach still need to be developed, the data required to investigate the conjecture, and the assumptions it relies on, have been determined.

Assumption 1:

One assumption which is useful when considering Progressive Refinement is that: A stable process exists which can be regarded as a series of stages, with the ratio of Effort in each stage being approximately constant between projects. (This is likely for an organisation which develops similar products. Where a new style of product is

developed, it is possible that the components will be similar to each other. The Effort ratios for the phases, for each component may then be similar.)

Given this assumption, the estimate could be refined when:

- A Development phase was completed and the Cost for that phase was known. For example, when the Requirements or Specification Phase was completed.
- When additional information became available. For example, when Function Point Counts or Lines of Code counts became available.
- When different builds of the software were completed. For example, at the end of an iteration for a product being developed using a Spiral Development Process.
- When phases in an evolutionary acquisition are completed.

The circumstances when the estimates could be refined depend on the process being used to develop the software and the desired rate of refinement. If Assumption 1 is violated, and the process is ad-hoc, then the estimates could only be refined when new deliverables were produced, or on a 'regular' basis. For example, estimates for testing and integrating the code could be refined after the Size (in Lines of Code) of the Source Code was known. If Function Point counts are derived earlier in the Software Development, they could also be used to refine the Cost of the project.

When a well-defined and controlled process is in place, the ratio of the Costs (Effort) for each stage of the Development should be similar between different Development projects. (A stage is a combination of Development phases and builds.) A hypothetical example, where the process has been broken into five stages, is shown in Figure 6. It shows the ratios a, b, c, d and e which could be obtained by normalising the total Effort. The curved line shown is the (hypothetical, normalised) true Effort over time. A step-wise approximation to this function is also shown. In practice, when Assumption 1 holds, the ratios could be determined from a number of previous projects.

Thus at the end of each phase or build, or both (depending on the process being used) the Cost could be re-estimated using: the Cost of the Development to date and, new information uncovered during the previous stage.

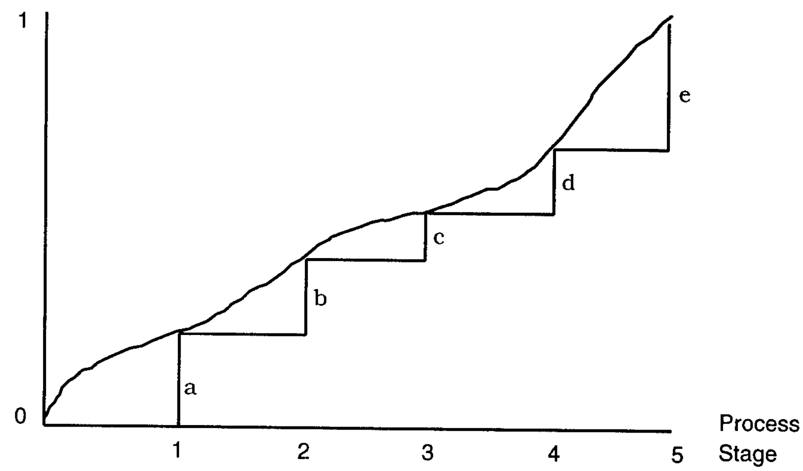

Figure 6: Ratios of process phases.

Figure 7 shows an example of how estimates could change over time when the Progressive Refinement approach is used.

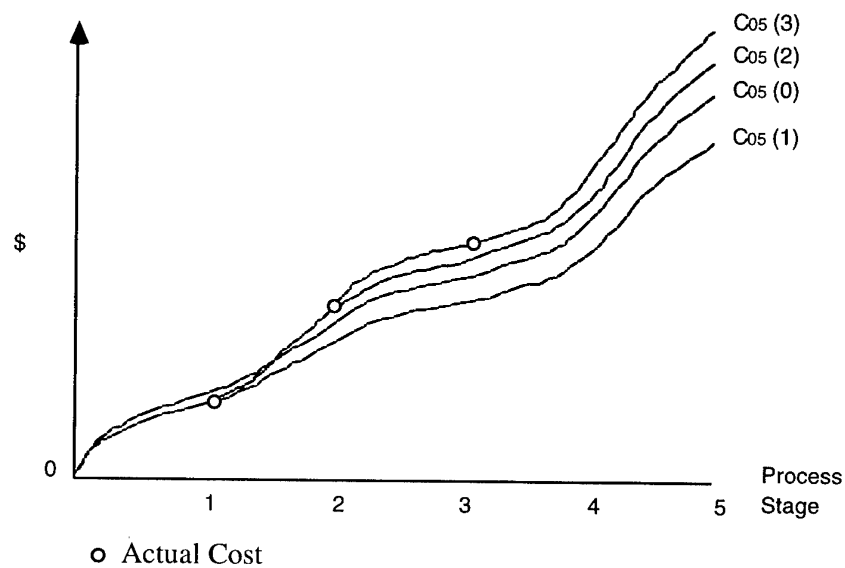

Figure 7: Refining Cost Estimates

One simple mechanism for determining new estimates from the actual Costs C_{ij} for each stage has been determined:

$$C_{05}(0) = \text{Initial Estimate}$$

$$C_{05}(1) = \frac{C_{01}(1) - D_{01}(1)}{a} + D_{01}(1) + D_{15}(1)$$

$$C_{05}(2) = \frac{C_{02}(2) - D_{02}(2)}{a+b} + D_{02}(2) + D_{25}(2)$$

where

- $C_{ij}(k)$ is the Cost for the completion of phase j including only Costs from the completion of phase i (i<j) and determined at the end of phase k. These are estimates where $k < j$ and they are observed values when $k \geq j$.

- D_{ij} is the deviation to the expected Cost for the period between phase i and phase j, which is due to known or anticipated deviations from the standard process. These are estimates where $k < j$ and they are observed values when $k \geq j$.

More complicated mechanisms which detect trends in these changes and allow the integration of new information, such as Function Point Counts will be investigated in the iMAPS and follow-on tasks.

4.6 Software Costing Using the SPRing Method

The SPRing approach is proposed as a method for determining the Effort, rather than the Cost required to develop a software system. However, the major Cost associated with software development are generally thought to be man-power costs.

For a given system, it is hypothesised that the development Effort could be reduced by improving the development environment. However, there would generally be costs associated with such improvements. For example, one method of improving the development environment is to train the development staff, and training can be expensive. In most circumstances, the total Cost of a system is more important than the Effort required to develop it [Crochow, 1995].

To compare the costs of developing the system in the two environments the Effort estimates for the environments should be calculated and then converted into Costs. In comparing the total Cost to develop the system in the two environments, the additional costs to change the environment should be included. However, as the new environment should also reduce the Effort required for subsequent systems, it might be preferable to spread these additional costs over future projects.

5. DISCUSSION

The two phase approach presented in this paper has the potential to substantially improve Software Costing and the risk management of software intensive projects.

Each of the concepts presented in this paper is currently being analysed and refined using measurement theoretic considerations. Each concept will be documented when it has been sufficiently refined.

Procedures are already in place to collect relevant data for the statistical analysis of the refined concepts, which will ensure that the method is suitable for use by the Australian Defence Industry.

The iMAPS task is due for completion in June 1996. Later tasks will continue the work started in the iMAPS tasks. The approach to be adopted in these tasks is intended to exploit the scientific progress and collaborative relationships made in the preceding iMAPS Task. It is also intended to interact closely with other ITD Tasks, notably AViDeS, JORN SE and AMD. Key characteristics of the approach to be adopted are:

- Scientific, ie repeated conjecture and attempted refutation.
- Empirical, ie experimentation based on real ADO data.
- Iterative, ie multiple cycles around a refinement loop.
- Robust, ie several technical threads run as a mix of parallel and serial activities.
- Collaborative, ie a core SE group team augmented by: (1) academic collaborators working as TSS contractors; (2) personnel from parts of ITD other than SE working on a part-time basis; (3) Defence staff from areas such as those of DSP-FDI, FASDM etc working at a low level of effort.

The tasks will focus on the following activities:

- Software Data collection - continuation and extension of the activities initiated in iMAPS.
- Software Development Costing - calibration, refinement and validation of the SPRing approach developed in iMAPS.
- Software Maintenance Costing - an extension of the SPRing approach.
- Software Risk Assessment - a theoretical framework to enable reasoning about Costing uncertainties.
- Policy formulation- influencing the formulation of Defence policy on Software Costing and Risk.
- Project/committee support - carefully limited support to Defence clients.

6. ACKNOWLEDGMENTS

The authors thank Stefan Landherr, Rudi Vernik and Peter Fisher of DSTO's Software Engineering Group, Professor Richard Jarrett and Andreas Kiermeier of the University

of Adelaide's Statistics Department and Professor Ross Jeffery of the University of New South Wales' School of Information Systems for their contributions in related discussions.

7. REFERENCES

J. W. Bailey and V. R. Basili, 1981 "A Meta-Model for Software Development Resource Expenditures." *Proceedings of the 5th International Conference on Software Engineering*: pp 107-116.

M. J. Baker and S. G. Eick , 1993 "Visualizing Software Systems". *Sixteenth International Conference on Software Engineering,* Baltimore MD, pp 59-67.

B. W. Boehm, 1984 "Software Engineering Economics." *IEEE Transactions on Software Engineering* SE-10(1): pp 4-21.

F. P. Brooks, 1987 "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer*(Apr): pp 10-19.

M. M. Burke, 1995 *iMAPS Task Plan - Issue 2* (No. DST 93/949), DSTO.

S. D. Conte et al., 1986 Software Engineering Metrics and Models. Menlo Park, Benjamin/Cummings.

J. M. Crochow, 1995 "Soft Solutions of Software Measurement." *IT Metrics Solutions.* **1**: pp 12-13.

S. G. Eick et al., 1992 "SeeSoft: Tool for Visualizing Line Oriented Software Statistics." *IEEE Transactions on Softwre Engineering* 18(11): pp 957-967.

F. J. Heemstra, 1992 "Software Cost Estimation." *Information and Software Technology* 34(10): pp 627-639.

D. R. Jeffery and G. Low, 1990 "Calibrating Estimation Tools for Software Development." *Software Engineering Journal* 5(July): pp 215-221.

Kemerer, 1991 "Software Cost Estimation Models." Software Engineer's Reference Book, Butterworth-Heinemann Ltd.

A. Kiermeier, 1994 *CEED Project: Project Proposal. Software Cost Prediction: A Statistical Approach.* (No. Project Number 94705), University of Adelaide.

G. Kingston, 1996 *iMAPS: Collecting Data for Software Costing,* Technical Report (No. DSTO-TR-XXXX. To be published), DSTO.

G. Kingston et al., 1996a *iMAPS: A Review of Productivity Factors for Software Cost Estimation,* General Document (To be published.), DSTO.

G. Kingston et al., 1996b *iMAPS: A Review of Software Sizing for Effort Estimation,* General Document (No. DSTO-GD-XXXX. To be published.), DSTO.

G. Kingston et al., 1995 "On the Statistical Significance of Productivity Factors in Software Development Effort Predicition.". *Australian Conference on Software Metrics,* Sydney, Australia, Australian Software Metrics Association, pp 179-191.

B. A. Kitchenham, 1992 "Empirical Studies of Assumptions that Underlie Software Cost-estimation Models." *Information and Software Technology* 34(4): pp 211-218.

J. E. Matson et al., 1994 "Software Development Cost Estimation Using Function Points." *IEEE Transactions on Software Engineering* 20(4): pp 275-286.

R. E. Park et al., 1994 *Software Cost and Schedule Estimating: A Process Improvement Initiative,* Special Report (No. CMU/SEI-94-SR-03), Software Engineering Institute, Carnagie-Mellon University.

R. Vernik, 1996 *Visualisation and Description in Software Engineering,* PhD, University of South Australia.

R. Vernik et al., 1993 "Description-Based Software Quality Evaluations". *Australian SoftWare Engineering Conference,* Sydney, Australia.

R. J. Vernik et al., 1991 "Automated Support for Assessment of Large Ada Software Systems". *TRI-Ada'91.*

The SPRing Approach to Software Costing

Gina Kingston and Martin Burke

(DSTO-TR-0460)

DISTRIBUTION LIST

	Number of Copies

AUSTRALIA

DEFENCE ORGANISATION

S&T Program	
Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Counsellor, Defence Science, London	Doc Control sheet
Counsellor, Defence Science, Washington	Doc Control sheet
Scientific Adviser to MRDC Thailand	Doc Control sheet
Director General Scientific Advisers and Trials)	1 shared copy
Scientific Adviser - Policy and Command)	
Director Science Policy - Force Development	1
Navy Scientific Adviser	3 copies of Doc Control sheet and 1 distribution list
Scientific Adviser - Army	Doc Control sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1

Aeronautical & Maritime Research Laboratory	
Director	1

Electronics and Surveillance Research Laboratory	
Director	1
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control sheet
Head, Information Architectures Group	1
Head, C3I Systems Engineering Group	1
Head, Information Warfare Studies Group	Doc Control sheet

Head, Software Engineering Group	1
Head, Trusted Computer Systems Group	1
Head, Advanced Computer Capabilities Group	Doc Control sheet
Head, Computer Systems Architecture Group	Doc Control sheet
Head, Systems Simulation and Assessment Group	Doc Control sheet
Head, Intelligence Systems Group	Doc Control sheet
Head, CCIS Interoperability Lab	Doc Control sheet
Head Command Support Systems Group	1
Head, C3I Operational Analysis Group	Doc Control sheet
Head Information Management and Fusion Group	Doc Control sheet
Head, Human Systems Integration Group	Doc Control sheet
Gina Kingston (Author)	3
Martin Burke (Author)	1
Publications and Publicity Officer, ITD	1
DSTO Library and Archives	
Library Fishermens Bend	1
Library Maribyrnong	1
Library DSTOS	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control sheet
Forces Executive	
Director General Force Development (Sea),	Doc Control sheet
Director General Force Development (Land),	Doc Control sheet
S&I Program	
Defence Intelligence Organisation	1
Library, Defence Signals Directorate	Doc Control sheet
Acquisition Program	
Assistant Secretary, Development Projects Management	1
Assistant Secretary, Joint Projects Management	1
Assistant Secretary, Project Planning and Evaluation	1
Assistant Secretary, Resources Planning - Major Capital Equipment	1
Director General, Information Management	1
Nulka Project Office - EO	1
B&M Program (libraries)	
TRS Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre (DEC),	1
US Defence Technical Information Center,	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1
Universities and Colleges	
Dr Barbara Kitchenham, Keele University	1
Professor Richard Jarrett, University of Adelaide	2

Australian Defence Force Academy	1
Library	1
Head of Aerospace and Mechanical Engineering	1
Deakin University, Serials Section (M list)), Deakin University Library,	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1

Other Organisations

NASA (Canberra)	1
AGPS	1
State Library of South Australia 1	
Parliamentary Library, South Australia (1	

OUTSIDE AUSTRALIA

Abstracting and Information Organisations

INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts	1
Documents Librarian, The Center for Research Libraries, US	1

Information Exchange Agreement Partners

Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1

SPARES	10
Total number of copies:	**72**

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
N/A

2. TITLE The SPRing Approach to Software Costing	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) Gina Kingston and Martin Burke	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-0460	AR-009-950	Technical Report	December 1996

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
N9505/13/3	93/349	DST	40	22

13. DOWNGRADING/DELIMITING INSTRUCTIONS N/A	14. RELEASE AUTHORITY Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS
Software costs
iMAPS
Software engineering

19. ABSTRACT
This paper describes a two-phase approach to Software Costing which has been proposed by the iMAPS Software Costing Research team. The proposed approach consists of obtaining an initial, rough, estimate of the Cost of the system during the first phase. During the second phase the estimate is refined throughout the development of the system as more information becomes available.

Page classification: UNCLASSIFIED